

Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Peter Ondrůška

Automatic assembly of jigsaw puzzles from digital images

Department of Software Engineering

Supervisor of the bachelor thesis: Mgr. Jiří Sedlář

Study programme: Computer Science

Specialization: General Computer Science

Prague 2011

UNIVERZITA KARLOVA
Matematicko-fyzikální fakulta

Katedra softwarového inženýrství

Akademický rok: 2010/2011

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jméno a příjmení: **Peter Ondrůška**

studijní program: **Informatika**

studijní obor: **obecná informatika**

Děkan fakulty Vám podle zákona č. 111/1998 Sb. určuje tuto bakalářskou práci:

Název práce:

Automatic assembly of jigsaw puzzles from digital images

Zásady pro vypracování:

The objective is a proposal and implementation of an algorithm for automatic assembly of jigsaw puzzle pieces from digital images. Problems to be solved include segmentation of the pieces, determination of their correspondence according to their shape and color pattern, and assembly of the puzzle that minimizes possible errors. The student should study journal articles for methods related to the problem and, based on this research, propose an innovative solution. The algorithm should be implemented in C++ and its performance demonstrated on images of jigsaw puzzles containing hundreds of pieces.

Seznam odborné literatury:

- [1] Pratt W. K.: *Digital Image Processing (3rd ed.)*, John Wiley, New York, 2001.
- [2] Gonzales R. C., Woods R. E., *Digital Image Processing (2nd ed.)*, Prentice Hall, 2002.
- [3] Zitová B., Flusser J., *Image registration methods: a survey. Image and Vision Computing*, 21 (2003), 11, pp. 977-1000.
- [4] Duda R.O. et al., *Pattern Classification, (2nd ed.)*, John Wiley, New York, 2001.
- [5] D. Goldberg, C. Malon, M. Bern: *A Global Approach to Automatic Solution of Jigsaw Puzzles. In Symposium on Computational Geometry*, 2002.
- [6] M.G. Chung, M Fleck and D.A. Forsyth: *Jigsaw Puzzle Solver Using Shape and Color. Proc. ICSP '98*, 1998, 877-880.

Books and journal articles on digital image processing and pattern recognition.


Vedoucí bakalářské práce: **Mgr. Sedlář Jiří**

Navrhovaní oponenti:

Konzultanti: **prof. Ing. Flusser Jan, DrSc.**
Mgr. Zitová Barbara, Ph.D.
Mgr. Mareš Martin, Ph.D.

Datum zadání bakalářské práce: **10.11.2010**

Termín odevzdání bakalářské práce: dle harmonogramu příslušného akademického roku


.....
Vedoucí katedry


.....
Děkan

V Praze dne 12.01.2011

I would like to thank my thesis advisor Mgr. Jiří Sedlář for his time, feedback and a lot of advice. I would also like to thank to my family and friends for their love and support.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In date

signature

Názov práce: Automatické skladanie puzzle z digitálnych snímok

Autor: Peter Ondrúška

Katedra: Katedra Softwarového Inžinýrství

Vedúci bakalárskej práce: Mgr. Jiří Sedlář

Abstrakt: V tejto práci popíšeme nový algoritmus na automatické skladanie klasických obrázkových puzzle pomocou počítača. Automatické skladanie zahŕňa spracovávanie obrázkov s naskenovanými dielcami, výpočet riešenia na základe vzájomnej kompatibility dielcov a vyprodukovania obrázku s výsledným riešením. Predchádzajúce metódy na riešenie tejto úlohy využívali len informáciu o tvare dielcov. Popísaný algoritmus využíva informáciu o tvare ale aj farbe dielcov a tiež prináša niekoľko zlepšení v rôznych aspektoch celého procesu. Testovanie nakoniec poukázalo, že táto metóda dosahuje lepšie výsledky ako všetky doterajšie algoritmy keď dokázala zložiť puzzle o veľkosti viac ako 1000 dielcov.

Kľúčové slová: puzzle, spracovávanie obrazu, rozoznávanie vzorov

Title: Automatic assembly of jigsaw puzzles from digital images

Author: Peter Ondrúška

Department: Department of Software Engineering

Supervisor: Mgr. Jiří Sedlář

Abstract: This thesis describes a new approach to automatic assembly of classical jigsaw puzzles by computer. The process involves processing scanned images of puzzle pieces, computing a solution based on piecewise compatibility and producing an image of the solution. Whereas previous approaches to this problem were mostly concentrated on using only the shape of the pieces, the method we proposed uses both shape and colour information. The method also introduced several improvements in different aspects of solving. The method was able to successfully solve a puzzle of more than 1000 pieces and thus outperformed previous algorithms.

Keywords: jigsaw puzzle, image processing, pattern recognition

Contents

1	Introduction	3
1.1	Problem formulation	3
1.2	Difficulties of the problem	4
2	Background	5
3	Method overview	7
4	Data extraction	8
4.1	Shape extraction from back scan	9
4.1.1	Format of images	10
4.1.2	Extraction method	10
4.2	Piece segmentation with known shape	10
4.2.1	Piece detection and assignment of shape	11
4.2.2	Matching shape to piece position	13
4.3	Edge classification	14
4.4	Colour extraction	16
5	Puzzle solving	17
5.1	Local compatibility	17
5.1.1	Filtering based on logical type	18
5.1.2	Optimal layout of compatible edges	18
5.1.3	Compatibility classification model	20
5.2	Frame assembling	22
5.2.1	Properties of the frame	22
5.2.2	Reduction to max-cost bipartite matching	23
5.3	Interior assembling	24
5.3.1	Greedy algorithm	25
6	Solution visualization	26
6.1	Computation of geometric configuration	26
6.1.1	Local position dependencies	27
6.1.2	Configuration based on dependencies	27
6.2	Image composition	28

7	Implementation and optimization	29
7.1	Parallelisation	29
7.2	Special algorithms	30
7.2.1	Shape optimization algorithm	30
7.2.2	Computing of compatibility table	31
8	Experimental results	32
8.1	Choice of testing data	32
8.2	Performance	33
8.2.1	Robustness	34
8.2.2	Shape-only case	34
8.2.3	Running time	34
8.2.4	Memory requirements	35
9	Discussion	36
10	Conclusion	37
	References	38
11	Attachment	40
11.1	Source code and testing data	40
11.2	Example input 1	41
11.3	Example input 2	42
11.4	Solution to a 80-piece puzzle	43
11.5	Solution to a 208-piece puzzle	44
11.6	Solution to a 572-piece puzzle	45
11.7	Solution to a 1008-piece puzzle	46

Chapter 1

Introduction

In this thesis we will introduce an efficient algorithm for automatically finding a solution to a standard jigsaw puzzle. A jigsaw puzzle is a common puzzle that consists of an image divided into many small pieces. Solving the jigsaw puzzle involves assembling these pieces together to form the original image.

An automatic solution to a jigsaw puzzle is a highly non trivial problem. It becomes even more complex if information about the desired solution, i.e. the original image, is not available. The existing algorithms are usually based on various techniques used in manual solving. Although several automatic methods solve the problem efficiently, none of them were designed to solve very large puzzles, i.e. more than a thousand pieces.

The automatic solving of jigsaw puzzles has many applications, namely in image reconstruction, computer vision, robotics, combinatorial optimization and other fields of applied mathematics. Existing automatic methods also provide insight into computer efficiency compared to classical manual solving.

1.1 Problem formulation

In this thesis, the term automatic solution to a jigsaw puzzle by computer will mean a process of solving the puzzle where only a minimum of human intelligence is required and most of the logic is handled by a computer.

The input of this process is a set of digitally scanned images of puzzle pieces. This is the only input; we do not use other information like the desired assembled image, which is usually available at the top of the jigsaw puzzle box.

The algorithm extracts the pieces from the input images and assembles an anticipated solution of the puzzle. The problem of puzzle assembling itself can be defined as an optimization problem of finding the configuration maximizing local compatibility of neighbouring pieces.

The output is an image of these pieces in a geometrical configuration corresponding to the anticipated solution. This image should be very similar to that of the manual solution (Figure 1.1).

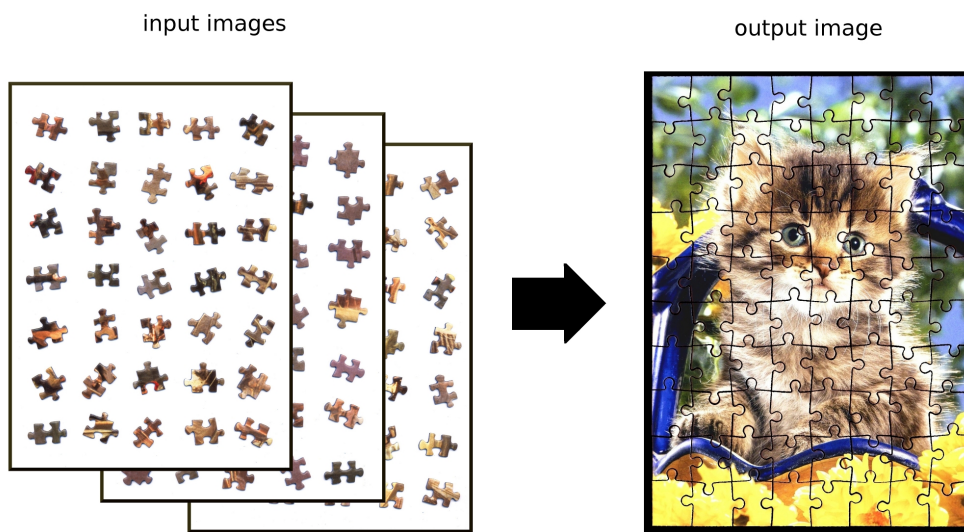


Figure 1.1: Automatic solving of puzzles.

Puzzle type specification

We will focus on jigsaw puzzles where each piece has four edges that can connect to other pieces by uniquely corresponding indents and tabs. This representation corresponds with a model where each piece is a square and the solution is a convenient placement of these squares in a rectangular grid. This abstraction is also known as an instance of square-tiling problem [11].

1.2 Difficulties of the problem

The problem of solving a jigsaw puzzle is not only theoretically intriguing but also technically challenging.

First of all the jigsaw digitalization process introduces several difficulties. Discrete input data obtained by a scanner usually contain several kinds of noise and specific types of error, e.g. shadows or reflections. This dramatically affects the quality of extracted data. For these reasons, very robust processing methods and techniques are necessary.

Moreover, the fact that we do not have information about the desired solution makes the problem very hard. In general, the problem of finding a solution to this type of puzzle is NP-complete [11]. This means that no efficient polynomial algorithm is known. This complexity has also led to production of puzzles with high rewards for finding solution [12].

Although the variability of pieces in shape and colour can help to determine matching edges in the solution, it is still very challenging to develop combinatorial method that can effectively use the noisy information

In conclusion, it is very hard to design a reliable algorithm that can handle jigsaw puzzles with a very large number of pieces.

Chapter 2

Background

Because of an interesting nature of the problem, many attempts of computer solving of real jigsaw puzzle were made.

One of the first results is from Freeman and Gardner [1] followed by many papers [2] [3] [4]. These earlier approaches were rather simple and used only information about shape of the pieces. However they were able to assemble the puzzles of size a few dozens of pieces.

The raising performance of modern computers and possibility to process more information allowed development of more sophisticated methods. The first attempt was to use each piece's colour together with shape from Chung, Fleck and Forsyth [5]. Their work also showed the new combinatorial approach of transforming the problem into an instance of the Travelling Salesman Problem [13]. As demonstrated, this method was able to solve the puzzle of size 54 pieces.

Later methods [6] [7] improved the quality of piece's shape extraction. Precise information of the shape was shown as key aspect during successful solving and allowed to develop methods able to solve puzzles of more than 100 pieces.

The project of Atsmon and Varon [8] from 2009 dramatically improved the technique of extracting the shape information. To obtain high precision was each piece scanned three times; twice from the front side, against the background of different colour, and once from the back side. Data obtained by this method was good enough to assemble the biggest puzzles yet of size 572 pieces.

The general nature of solving a jigsaw puzzle also helped the development of methods of other connected problems. One of them is solving a puzzle where each piece is a square which implies we can not use any shape information. The best result in this area is from Cho, Avidan and Freeman [9] whose probabilistic algorithm was able to solve puzzle of size 400 pieces. We must note, this puzzles were obtained synthetically, by digital dividing of the picture into squares. Data of this type does not contain many kinds of noise present in the extracted colour from digitalised real puzzle.

Finally, one of the curious methods of piece-wise compatibility checking used the "mechanic oraculum"[10]. This oraculum was able to physically check if the given pieces were really shape compatible.

To sum up, many previous approaches were made to solve the defined problem with various results. Most of them use only the shape information of the pieces and can be improved in several ways to possibly achieve better results.

Chapter 3

Method overview

Our proposed method for solving of jigsaw puzzles is improving techniques used in prior approaches discussed in the background chapter and also inventing some new. The method is primarily designed to be able successfully and effectively solve the puzzles of sizes which were not possible to solve before. The intended performance was also confirmed during testing when the method was able to solve the puzzle of size more than thousand of pieces beating the former best result in this area. The method itself consists from several consecutive steps resulting to final solved puzzles.

The input data consists of images obtained by scanning puzzle pieces on a standard scanner. To improve the precision of piece features extraction, we use a similar approach as in [8]. This involves scanning of the pieces from both the front and the back side (Figure 11.1, 11.2), but with a difference that we need to scan the front side only once.

Extracting the pieces from the images requires a multi-stage process. The shape of pieces is for higher quality extracted separately from the images of their back sides. This shape is later mapped back to the images of pieces scanned from the front side. The colour and other features required during later puzzle solving are also extracted during the preprocessing.

The assembling of the puzzle is due to a high complexity of the problem done using combined heuristic algorithm similar to work [5]. We assemble the puzzle in two steps: assembling a frame and assembling an interior of the frame.

To solve the frame, we use reduction of the problem into a restricted case of a max-cost bipartite matching [15]. Solving the interior involves a greedy algorithm with limited backtracking.

Finally, the obtained combinatoric solution is transformed into a desired output image. On this image are the input pieces in a configuration forming the expected solution of the puzzle.

Chapter 4

Data extraction

Data extraction means extraction of parameters of each puzzle piece from the input images. This information contains each piece's shape and colour along its edges. We use this information to create an internal representation of each piece.

Piece is internally represented by four edges which can be used to connect another pieces. For each edge we store its shape as a sequence of points along particular part of piece's boundary. Moreover, for each this point we store an average colour of the piece in its neighbourhood.

High precision extraction of puzzle pieces

The extracting of the pieces' parameters is not an easy task. Simple methods often do not provide sufficient results and more sophisticated approaches are required.

First, we tried a method of scanning puzzle pieces using a standard image scanner, we placed the pieces at the scanner and scanned them against a contrast background. Despite using various scanners and scanning in a high resolution, this method did not provide sufficient precision. This was caused by three main errors shown on the Figure 4:

1. The shadow around the pieces caused by the scanner's light.
2. Capturing parts of pieces which are not visible in an assembled puzzle.
3. Similarity of piece's colour to background colour.

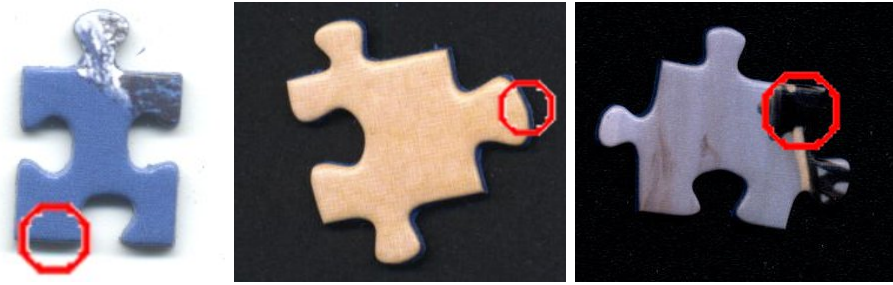


Figure 4.1: Three main errors of scanned pieces.

To overcome these limitations, we use a more sophisticated approach. We found that scanning puzzle pieces from the back side does not contain any of mentioned errors. Although this does not provide any information about colour of pieces, it allows us to precisely extract their shape. Knowledge of the shape can be then used in a significant improvement of the image segmentation in the previously described method.

To apply this observation we scan each piece two times. First, we scan the puzzles as described in the first approach (Figure 11.1). For each front scan we also create a corresponding back scan. In the back scan, we simply flip all the pieces on the scanner and make a picture with back sides of each piece (Figure 11.2). For correct identification of corresponding pieces we need to create this back scan in a configuration, when each piece is approximately at the same location as in the original scan. It is also critical to scan the images in the same resolution to have the same scale for all pieces.

The extracted shapes from the back scan are then used during the processing of original images when the problem is reduced to image segmentation into objects with a known shape.

After the segmentation we split the shape of each piece into four parts corresponding to its four edges.

Finally, extracting of the colour along the edges, we obtain the desired internal representation of the pieces.

The described multistage approach of extracting the shape and colour information of the pieces from different images produces the data with very high quality.

4.1 Shape extraction from back scan

As described, shape extraction of the puzzle pieces is executed on the images of their scanned back sides. This images does not contain any significant errors affecting the border of puzzle pieces allowing us to obtain very precise results.

4.1.1 Format of images

Input images should be created in a way that allows the best image segmentation. This is achieved when the usually bright back sides of pieces are scanned against a dark background. Pieces should also not overlap or touch to correctly detect their shapes.

4.1.2 Extraction method

The shape extraction of puzzle pieces on defined images is a segmentation of bright objects from a dark background. To solve this problem, we use a classical method of transforming the input images into a grayscale, performing a threshold-based segmentation and extracting the shape of segmented objects.

To transform the image into grayscale we can use one of many standard methods. Testing showed a very good method is to set gray value of a particular pixel to the maximum value of its red, green and blue colour components.

The following segmentation of this image is obtained simply by considering all the pixels with value greater than given threshold as the foreground and all other pixels as the background. To automatically calculate the proper threshold value we can use the classical Otsu's histogram shape-based method [18].

For the correct results, it is necessary to further filter noisy foreground components. The area of each piece is approximately the same, we can therefore discard all the components with the area less than $\frac{1}{4}$ of the greatest component's size. Using a morphological operation close [19] with a circle operator of the size a few pixels can further reduce the noise along the border of the components.

The desired piece shapes are finally extracted as sequences of border pixels' coordinates of the foreground components.

4.2 Piece segmentation with known shape

Piece segmentation from the images is performed after we extract their shape from the corresponding back scan. The knowledge of the shape reduces the problem to position calculation for each this shape to match the desired piece and segmentation along this shape. We propose a effective method for calculating of this positions for each known shape.

First we detect approximate position of each piece and to every this piece we assign one known shape. Then we find exact position of each shape to match the assigned piece on the image.

Finally we simply cut the pieces from the image based on this placed shape obtaining the desired.

Format of images

To successfully apply the method, the input images should be created in a way that allows good detection of puzzle pieces. For this reason it is enough that we scan pieces with mostly bright texture against a dark background and vice versa (Figure 11.1, 11.3). Moreover, as mentioned for determining the correspondence of the pieces to the known shapes, corresponding pieces has to be approximately at the same position in the front and the back scan.

4.2.1 Piece detection and assignment of shape

For the further processing we need for each piece on the image detect its position and assign one of known shapes. Detection is based on a detection of objects with texture different from the background. For each such detected piece we also assign its shape based on the similarity of piece's position on the front and the back scan.

At the beginning, we detect the foreground objects using simple segmentation technique based on the similarity to an average background colour. The average background colour can be computed as an average colour of border pixels because this pixels should not be parts of the pieces. Then we determine for each pixel if it is a part of foreground or background based on the colour distance to to this computed colour.

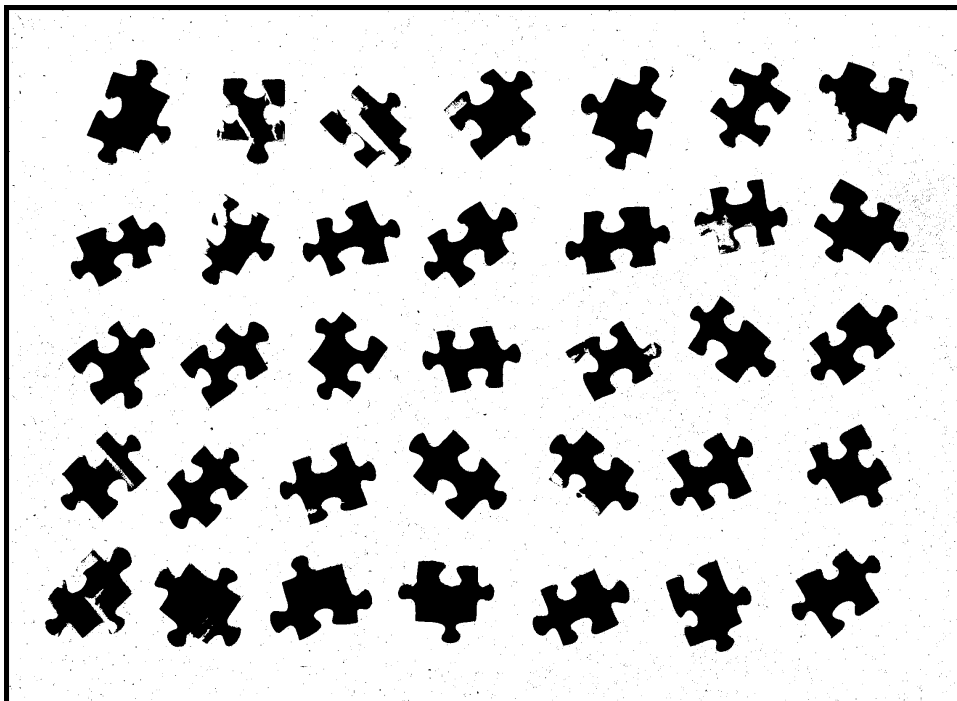


Figure 4.2: Binarized image with puzzle pieces.

Applying this technique we obtain a binarized image with pieces forming the foreground. However this image can contain various errors like noise pixels or a division of one piece into two components (Figure 4.2).

Next, we determine for each foreground pixel its correspondence to some piece. Analysis shows, the shape of each piece has approximately the same variance in each direction. We can use K-means clustering [17] to determine this correspondences.

As a feature space we use directly the coordinates of foreground pixels. Because the position of each piece on the picture is approximately the same compared to the corresponding back scan, we also set the initial means to the centres of masses of extracted shapes from the back scan. After clustering, we obtain clusters corresponding with each piece on the image 4.3.

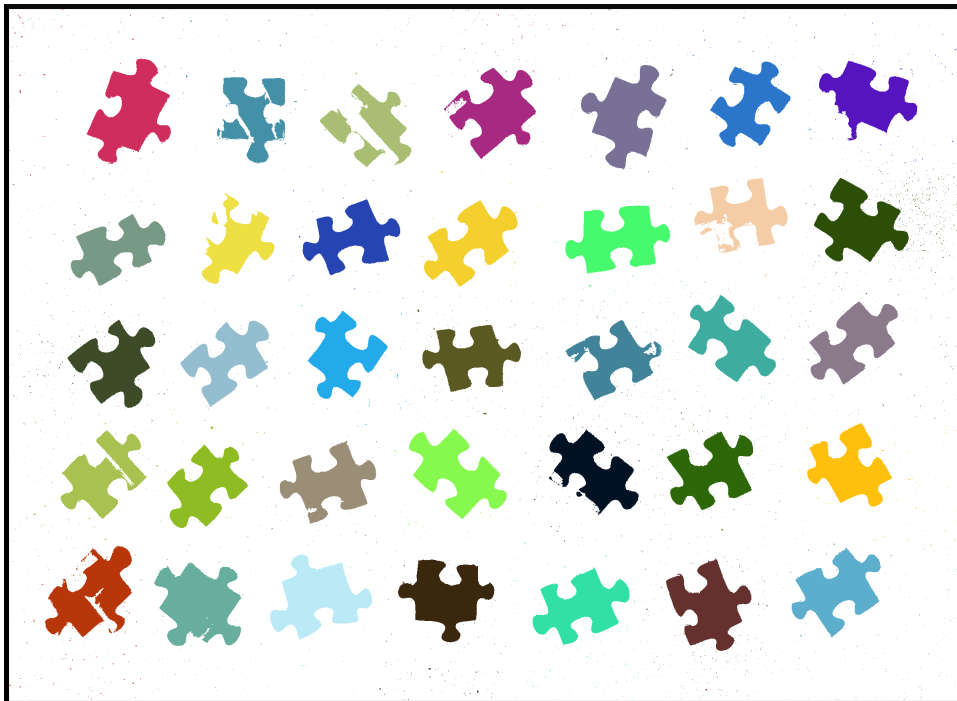


Figure 4.3: Clusterized image using K-means clustering.

However the custerizing can theoretically determine wrong clusters, in practice this methods works very well. As it can be seen, also the pieces binarized into several components are assigned to one cluster.

This clustering also automatically determines the assignment of the shapes to the pieces. We simply assign to each cluster the shape whose center of mass was originally equal to the mean of the cluster.

4.2.2 Matching shape to piece position

For each detected piece and its assigned shape we find the exact mapping of the shape to match the border of the piece. This mapping can be described as a translation and rotation of the shape; transformation also known as rigid. The computing of this transformation requires a special algorithm.

Because of the image format, we can assume, the pieces on the images have mostly recognizable edges. The edges can be found applying some well-known filters on the image, for our purposes the Laplace operator[23] works well.

We can therefore define the optimal position for each shape as one matching most of the edge pixels. However, this optimal position do not have to correctly match the shape of the piece, testing never shown the opposite.

One of the methods which can find the optimal transformation is the Extended Hough Transform [21]. Unfortunately, computational complexity of this method makes its real use for our purposes impossible.

To overcome this difficulty, we need to use a different approach. First we place the shape to a base position when the center of the shape's mass is equal to the center of mass of the piece (the mean of its cluster). Then we rotate the shape by 5 degrees and after each rotation we run a Shape Optimization Algorithm to find the exact transformation of the shape to the piece from this base position. Finally we choose the transformation which gives best score returned by the algorithm.

Using of this two-step method of placing the shape at the approximately correct position and optimizing the exact transformation we obtain very precise results comparable to using of the Hough Transform but with dramatically less computation.

Shape Optimization Algorithm

After we place the shape at approximately correct position we need to find a locally best matching of the shape to the piece. We use an iterative algorithm which in each iteration reoptimizes the current transformation to better match underlying edges of the piece.

The essential aspect is using of the Schwartz-Sharir algorithm[22]. This algorithm accepts two sets of points in the plane; for every point from the first set we have defined its matching point in the second set. The algorithm finds the optimal rigid transformation of the first set minimizing the sum of distance squares to all matching points in the second set.

In each iteration of the algorithm we find for each point on the shape the average position of the edge within its local neighbourhood. We use the square neighbourhood having the center in the given point. The average position of the edge can be defined as the center of mass of this neighbourhood when

each pixel has the value equal to the value of the edge assigned by the edge detection.

Next we use this pairs of points as the input of the Shwartz-Sharir algorithm to find the optimal transformation of the shape to match the average positions of the edge in the local neighbourhood of the shape.

After each iteration the transformation of the shape better matches the underlying edge. As the transformation converges to the local optimum, we can also slightly decrease the size of the neighbourhood in each iteration. We stop when the transformation is not changing or the change is under given ϵ .

Finally, for a purpose of decision which converged transformation from the 72 basic positions, rotated by 5 degrees, is the best we also output the score of the transformation found. The score is defined as the sum of values of the pixels in the constant neighbourhood of each point on the shape. The transformation with high score indicates the strong underlying edge close to the shape position.

Testing showed, the proposed algorithm gives very good results when the shape precisely match the edges of the given piece. The algorithm converges usually very fast within about 10 iterations. Moreover, the chapter 7 discuss a method how the average position of the edge can be computed in a constant time further improving the running time of the method.

4.3 Edge classification

In order to obtain an internal representation, we need to split each shape into four parts representing the edges of the piece. basic goal is therefore to identify the four corners which define this four edges.

However it is rather easy to develop an arbitrary algorithm, the challenge is to develop an algorithm having success rate of over 99.9%. Incorrectly detected edges of one piece can later cause a failure of the entire puzzle solving.

The detection is based on the first and second derivative of piece's shape shown on Figure 4.4. The corners of the pieces are points with high curvature which can be seen as peaks in second difference. As it can be seen, this peaks do not have to be the global maximum.

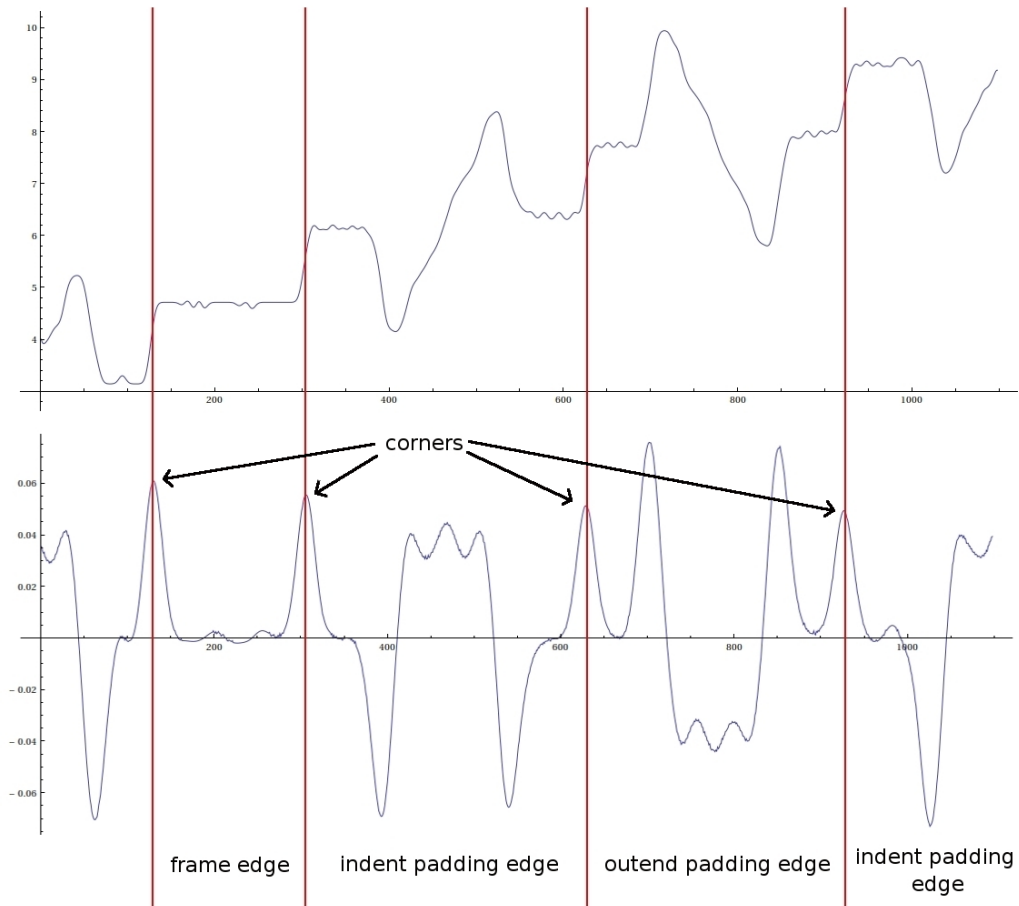


Figure 4.4: First and second derivation of the piece's shape.

In our method we first identify all candidates for corner points. Testing on various pieces showed, the correct corners are always local maxima on the second derivative with value greater than one third of the global maximum. This restriction usually satisfy 4-12 points on the curve. The solution is then a combination of four points from this set.

To select the proper combination, we assign a penalisation score for each possible combination and choose one with lowest score. The penalisation score is based on the sum of penalisation scores for subcurves defined by the given combination of points.

The penalisation score for the subcurve is a score reflecting a degree of similarity to some classical type of edge - edge with indent, outdent or no padding. For each curve we compute three different scores measuring its similarity to each of type. At the end, we choose the lowest score which also identifies the type of the edge.

The first core measuring similarity to an edge without padding can be computed rather easily. It is enough to simply compute the sum of distances

of points forming the curve to a virtual line between the first and the last point.

The score measuring the similarity to edge with outdent padding is based on a significant valley in the second derivative of the curve. This valley is caused by padding of the edge. To compute the score, we delete this valley part of the curve, which is equivalent to cutting out the padding part, and measure the distance to edge without padding.

Equivalently we compute the similarity to edge without padding. In this case it is enough to flip the edge and measure the similarity to the edge with the outdent padding.

Despite higher complexity of the proposed method, this approach of analysing each possibility provides highly reliable method of identifying the piece's corners.

4.4 Colour extraction

In order to use colour information during the puzzle solving, it is required to extract this information along each edge. Basically, for each point forming the edge we want to store a colour in a neighbourhood of this point.

Because the colour information on the edges can be damaged by image artefacts like light reflections we extract the colour of the piece in a constant distance from the edge. Proper distance is dependent on quality and a resolution of input data. For inputs with resolution 100 dpi it is usually enough to use a distance of 3 pixels.

Finally, for purposes of noise reduction, it is encouraged to blur the image using Gaussian filter[20] with low sigma before the extraction. This will improve the quality of the obtained information by considering a wider area around the extraction point.

Chapter 5

Puzzle solving

In this chapter we describe the algorithm assembling the extracted puzzles into a final solution. The solving of the puzzles involves several aspects.

The solving is done on a simplified combinatorial model of the puzzles. In this model, every piece is considered a square with four edges. The combinatorial solution of this puzzle is a suitable placement of these squares into a grid creating the solution of the puzzle. It means, for every piece we need to compute its position in the grid and one of its four possible rotations.

Because we do not have any information about the desired solution, the key aspect during the solving is the local compatibility of neighbouring pieces. Generally we want to measure the likelihood of coincidence of two edges in the correct solution which can be expressed by their similarity score, and find a solution which maximizes this likelihood.

The final solving of puzzles is a very complex combinatorial task. We use a combined heuristic search with limited backtracking to find the solution in a reasonable time. We use approach similar to algorithm used in the work [8] - solving the frame of the puzzle first and then filling the interior of this frame.

The entire solving process finishes with the combinatorial solution of the puzzle. This solution is later used during the visualization phase.

5.1 Local compatibility

The measuring of local compatibility of two edges is based on several factors. Logical incompatibility of some pairs of the edges can be decided rather easily. For the rest of the edges we have to use a method allowing us to measure their relative compatibility in some more continuous way to deal with a lot of possible combinations.

For the pairs of edges where the logical type does not provide a valuable information, we measure their relative compatibility score $P(i, j)$. This score is proportional to a likelihood of incidence of given edges in a correctly solved puzzles.

To compute the compatibility score of two edges, we use a different approach from most of previous work. Instead of a direct measurement of a vector similarity describing particular edges, we measure a compatibility of an optimal geometric placement of two edges.

The compatibility of optimal geometric placement is described by an absolute shape and colour similarity score. Later we use a compatibility classification model transforming these scores into final single score $P(i, j)$.

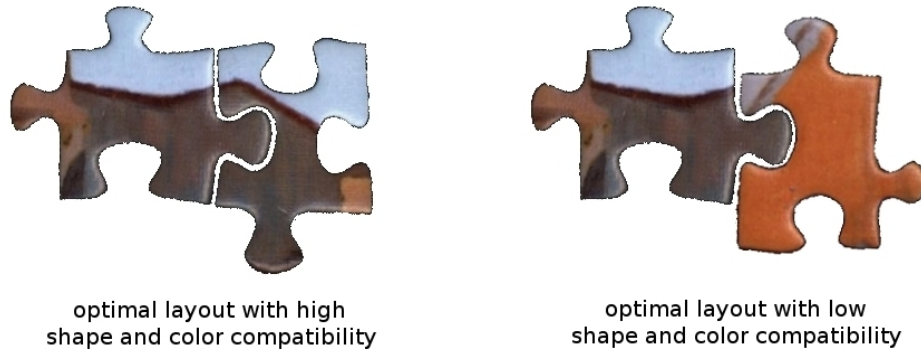


Figure 5.1: Optimal geometric layout of different pieces.

5.1.1 Filtering based on logical type

In general, the shape of each edge defines a logical type of the edge - the edge with indent, outdent or no padding. Some combinations can not logically fit together. For example, an edge with the indent padding has to coincide only with an edge having the outdent padding. The edges with no padding are located on the frame of the puzzle and can not coincide with any other edge. Observation that a piece containing an edge with no padding can coincide only with some other types of pieces allows the filtering of some more combinations.

5.1.2 Optimal layout of compatible edges

The geometric layout of two edges is some placement of one edge along the other. We introduce a method of finding this layout which is optimal in some matter.

Each placement can be encoded as a rigid transformation of the second edge when the position of the first edge is fixed. During the analysis of optimal geometric layout we work with defined internal shape representation of edges as a sequence of 2D points along their shapes. Therefore, fixing of the first edge's position means fixing all its points a_i ; rigid transformation of the second edge is applying this transformation to all its points b_j to obtain a new sequence b'_j

For a given geometric layout we define pairs of matching points. These pairs are useful during the analysis of correctly matching edges, because they should have similar characteristics.

Matching points

For the given geometric layout we define a matching point $N_{B'}(a_i)$ for a point a_i located on the first edge as the closest point from the set B' of points forming the second edge. To determine the closest point we use the standard euclidean metrics. Similarly we define $N_A(b'_j)$ as the closest point on the first edge to the given point b'_j located on the second edge.

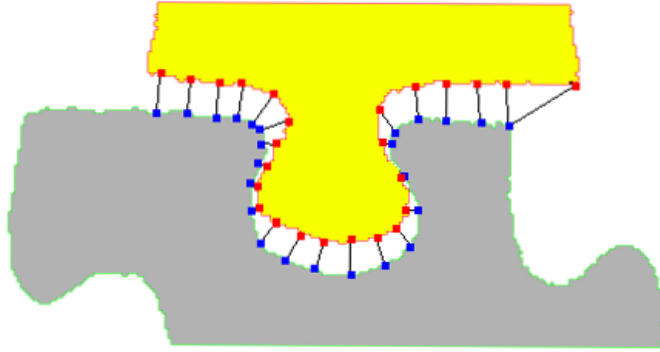


Figure 5.2: Matching points under given geometric layout

Optimal transformation

The optimal transformation for two edges should place these edges in a visually smooth layout when they best fit together. To judge different transformations, we define their shape compatibility score.

The shape compatibility score of given transformation is the sum of squared distances to all matching points:

$$K(i, j) = \sum_i \|a_i - N_{B'}(a_i)\|^2 + \sum_j \|b'_j - N_A(b'_j)\|^2 \quad (5.1)$$

As the optimal transformation we therefore consider one minimizing this shape compatibility score.

To compute the optimal transformation, we use an iterative algorithm similar to the method used in section 4.1.2. This algorithm gradually reoptimizes the actual configuration converging to a local optimum. This optimum can be the global optimum or is very close to this global optimum allowing us to use the result during further processing.

We start in a layout when we roughly place two edges together. This configuration can be found for example considering only the first and the last point on each edge.

In every iteration we use the Schwartz-Sharir algorithm[22] to reoptimize the transformation. First, we find all matching points for given transformation. Then we use this pairs of points as an input of Schwartz-Sharir algorithm to find an optimal transformation which minimizes shape compatibility score under given assignment. Finally we apply this transformation changing the actual geometric layout and also matching points. Moreover, each iteration will necessarily decrease or not change the overall shape compatibility score.

We iterate an algorithm until the shape compatibility score is not changing or the change is less than given ϵ . Testing showed, that proposed algorithm converges very fast in about 10 iterations and the found solution is very close to the global optimum.

5.1.3 Compatibility classification model

We measure the similarity of the edges in the optimal geometric layout and compute the compatibility score based on this similarity and other factors.

The similarity is of two types: the shape similarity and the colour similarity. Both of them can be described by a corresponding score. The score computation is based on the matching points which should have similar characteristics in the case of compatible edges.

Next, this scores are normalized to express the relative compatibility compared to the best possibility for the given edges and combined together to obtain the single final score.

The entire process judges the given layout of edges in terms of likelihood to be achieved by really matching edges. This likelihood is eventually expressed by the score later used to find the correct solution.

Shape compatibility score

We define the shape score directly as the defined shape compatibility score of the given layout normalized by the number of points on edges:

$$S(i, j) = \frac{1}{|A| + |B'|} \left(\sum_i \|a_i - N_{B'}(a_i)\|^2 + \sum_j \|b'_j - N_A(b'_j)\|^2 \right) \quad (5.2)$$

Colour compatibility score

The colour score consists of three separate scores of colour similarity of the matching points in the colour model HSL [23]. This model is used because it can better express the importance of the different colour properties.

$$C_H(i, j) = \frac{1}{|A| + |B'|} \left(\sum_i |a_{iH} - N_{B'}(a_i)_H|^2 + \sum_j |b'_{jH} - N_A(b'_j)_H|^2 \right) \quad (5.3)$$

$$C_S(i, j) = \frac{1}{|A| + |B'|} \left(\sum_i |a_{iS} - N_{B'}(a_i)_S|^2 + \sum_j |b'_{jS} - N_A(b'_j)_S|^2 \right) \quad (5.4)$$

$$C_L(i, j) = \frac{1}{|A| + |B'|} \left(\sum_i |a_{iL} - N_{B'}(a_i)_L|^2 + \sum_j |b'_{jL} - N_A(b'_j)_L|^2 \right) \quad (5.5)$$

Note, computation of $C_H(i, j)$ requires using the metrics measuring distance of points on a circle with a length of the perimeter 1.0.

Combining scores

The measured shape and colour scores express the absolute compatibility of the given edges and does not provide the information if the edges are really matching or not. To accommodate this property, we normalise each score with the best achieved score for the given edge:

$$S'(i, j) = \frac{\min\{S(i, k) | \forall k\}}{S(i, j)} \quad (5.6)$$

$$C'_H(i, j) = \frac{\min\{C_H(i, k) | \forall k\}}{C_H(i, j)} \quad (5.7)$$

$$C'_S(i, j) = \frac{\min\{C_S(i, k) | \forall k\}}{C_S(i, j)} \quad (5.8)$$

$$C'_L(i, j) = \frac{\min\{C_L(i, k) | \forall k\}}{C_L(i, j)} \quad (5.9)$$

Note, the best value of the score for compatible edges is 1.0 and also in general $S'(i, j) \neq S'(j, i)$, similarly for the other scores. To obtain the commutative property, we work with modified scores:

$$S''(i, j) = S'(i, j) + S'(j, i) \quad (5.10)$$

$$C''_H(i, j) = C'_H(i, j) + C'_H(j, i) \quad (5.11)$$

$$C''_S(i, j) = C'_S(i, j) + C'_S(j, i) \quad (5.12)$$

$$C''_L(i, j) = C'_L(i, j) + C'_L(j, i) \quad (5.13)$$

The final compatibility score is obtained simply by a linear combination of the defined scores:

$$P(i, j) = \alpha S''(i, j) + \beta C''_H(i, j) + \gamma C''_S(i, j) + \delta C''_L(i, j) \quad (5.14)$$

The values of the coefficients were obtained by testing on the training set of 4000 edges with known matching edges pairs in a correct solution. This created the 4000 positive and 15996000 negative cases of matching edges. We were looking for the values maximizing the number of positive cases judged for given edge as the highest scoring possibility among the other possibilities. The values of the coefficients were estimated as $\alpha = 5, \beta = 1, \gamma = 5, \delta = 1$.

5.2 Frame assembling

Assembling the puzzle frame is defining the positions of the border pieces e.g. containing an edge without padding (Figure 5.3). This partial solution is used as the basic of the desired solution into which are inserted all other pieces. First we analyse the problem of frame solving and then we reduce this problem to a special case of another problem known as min-cost matching[?].

piece 7	piece 15	piece 37	piece 23	piece 4	piece 12	piece 38	piece 28
piece 18	empty slot	empty slot	empty slot	empty slot	empty slot	empty slot	piece 31
piece 6	empty slot	empty slot	empty slot	empty slot	empty slot	empty slot	piece 24
piece 11	empty slot	empty slot	empty slot	empty slot	empty slot	empty slot	piece 29
piece 14	piece 33	piece 25	piece 34	piece 39	piece 17	piece 2	piece 6

Figure 5.3: Puzzle configuration with solved frame.

5.2.1 Properties of the frame

The count of the border pieces, containing a frame edge, m together with the count of all pieces n uniquely defines the width and height of the solution. These values provide an essential information about the expected solution of the puzzle and can be easily found as the solution of the equations:

$$width = \frac{1}{4}(4 + m + \sqrt{16 + 8m + m^2 - 16n}) \quad (5.15)$$

$$height = \frac{1}{4}(4 + m - \sqrt{16 + 8m + m^2 - 16n}) \quad (5.16)$$

note, we can rotate the solution by 90 degrees by swapping the equations.

For each border piece we define its left and right matching edge. Left edge of the border piece is the preceding the edge without padding. Similarly right edge of the border piece is succeeding the edge without padding.

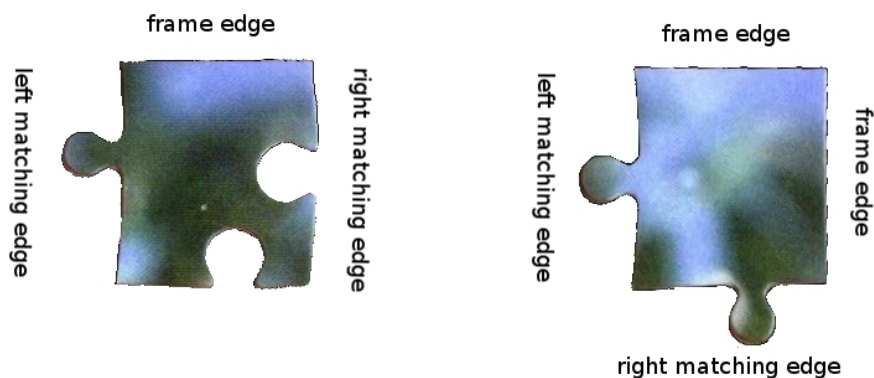


Figure 5.4: Matching edges of frame pieces.

In any correct solution, those edges are coincident with right respective left edges of another border pieces. In other words, it defines a perfect matching in a complete bipartite graph [14] with partitions represented by left and right edge.

5.2.2 Reduction to max-cost bipartite matching

The fact, each solution of the frame forms some matching in the given bipartite graph allows us to look at the problem from an opposite direction. We look for the matching defining the correct solution. This reverse process involves solution of more restrictive problem than simply finding any bipartite matching in the graph.

Not each perfect matching in this graph defines some correct solution. The matching has to fulfil two conditions to be considered a correct matching. The sequence of the left and the corresponding right matching edges has to create one closed cycle. Moreover, the relative distance of four corner pieces on this cycle has to match with expected dimensions of the puzzle.

We also define the optimal matching from the set of all correct matchings which is directly related to a solution of restricted max cost matching in the modified graph. This solution also defines the best solution of the frame.

Finally we propose the method to find this matching in a reasonable time.

Optimal matching

As an optimally assembled frame we consider one whose sum of compatibility scores $P(i, j)$ of incident edges is maximal. In other words, if we define the cost of the edges in the bipartite graph between vertices i, j as $P(i, j)$ (Figure 5.2.2), we will look for the correct matching with maximum cost[15]. This optimal frame does not have to be the intended one for the given puzzle, but the testing has never given an incorrect frame.

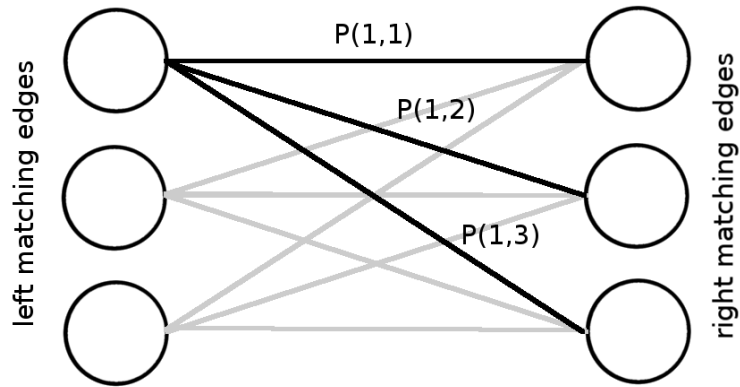


Figure 5.5: Bipartite graph with left and right edges.

Algorithm for optimal matching

Into described problem of finding correct max-cost matching can be easily transformed any instance of Travelling Salesman Problem (TSP) [13]. Because TSP itself is in a computational complexity class NP-hard, it implies there is no known polynomial algorithm for finding the correct max-cost perfect matching. However this fact an algorithm which can usually find the desired solution very fast can be used.

To find the exact solution we use similar approach as in work of [5]. We will gradually generate all perfect matchings in given bipartite graph in order of decreasing cost. After we generate each new matching, we check if it fulfils the conditions for the correctness. We stop after the first matching pass given conditions. It also implies this solution have the maximum cost, therefore is optimal and corresponds to the optimally assembled frame.

Algorithm for efficient generating permutations in given order is known as Murphy's algorithm[16] and can generate each next matching in polynomial time $T(m) = O(m^3)$. The overall complexity is therefore dependant on the number of generated permutations which can be in the worst case $O(m!T(m))$. However the testing shows this algorithm can usually find the solution on given data very fast generating only a few matchings. Often even the first matching passes the given conditions.

5.3 Interior assembling

After we have correctly solved the frame of the puzzle, we fill its interior to obtain the final combinatorial solution. Because of a huge number of possibilities of assembling the interior we use a greedy algorithm with limited backtracking to find the correct one in reasonable time. This algorithm does

not have to always find the correct solution, but during testing it was always able to find the solution of even very large puzzles.

5.3.1 Greedy algorithm

The greedy algorithm is based on placing piece at some empty position one at a time, the key assumption of correctly solved puzzles is therefore a correct placing of each piece in every step otherwise we obtain a wrong solution. For this reason we need to maximize the probability of correctness of each step.

To obtain the maximum likelihood of correct step, we always place some unused piece at some position which coincide with maximal number of already placed pieces. This number is always at least 2 and at the beginning we have four of this possible positions in the corners of the frame. Moreover we always choose the possibility of maximizing the sum of scores $P(i, j)$ to already placed edges.

We can further improve the likelihood of particular step correctness with backtracking of its neighbourhood. It means, for each possibility we also find the optimal filling of its empty neighbouring positions and similarly add their scores. In this case we also need to normalize each score by the number of included edges to not discriminate some possibilities.

Finally, during the assembling, we must not forget about the fact that scores $P(i, j)$ are always relativised to actual best possible matching for a given edge. Because these possibilities are changing during the solving, we also need to keep these scores accurate.

In conclusion, successfulness of the proposed greedy algorithm is highly dependant on the correctness of each single step. However, there is still a probability of finding a wrong solution, stated improvements greatly eliminated this cases on tested puzzles.

Chapter 6

Solution visualization

This chapter describes the process of visualization of the puzzle's combinatoric solution obtained during the puzzle solving phase into a final image with solved puzzle. The final image consists of pieces extracted during the data extraction phase in a configuration forming the desired solution. The computation of the configuration involves several aspects.

The basic goal is to compute the size of the resulting image in pixels w, h and for each piece its position in this image. The position of each piece can be described again as a rigid transformation of the piece defining its center coordinates x_i, y_i and a rotation ω_i .

The computed geometric configuration is then transformed into the final image. This is basically achieved by applying the computed transformations on the pieces and fusion into the resulting image.

6.1 Computation of geometric configuration

The computation of the geometric configuration involves computing the defined parameters. To obtain a visually pleasant result, the desired configuration should minimize the inconsistency between neighbouring pieces in the final image. We use the special method to compute the configuration having this property.

First, we compute the local position dependencies between the neighbouring pieces. This dependencies are stating the ideal relative transformation of one piece based on the transformation of another piece. Similarly we compute the position dependencies between the pieces with a frame edge and the frame.

Second, we express this dependencies as equations of a overdetermined linear system. Variables of the system are the required parameters of the final configuration and the equations are the dependencies of the values.

Finally, we use the least squares method [24] to find the solution of the defined overdetermined linear system.

Using the proposed algorithm of transforming the problem into the overdetermined linear system and finding the solution using the least squares method,

we obtain the configuration resulting into a very consistent result.

6.1.1 Local position dependencies

The local position dependencies are stating the optimal dependencies between some values of x_i, y_i, w, h, ω_i in the resulting solution. This dependencies are of two types: dependencies between the parameters of coincident pieces and the dependencies between pieces having an edge without padding and the frame of the puzzle.

The optimal local position dependency between the neighbouring pieces is the relative value of rotation $\Delta\omega_{ij}$ and the center position $\Delta x_{ij}, \Delta y_{ij}$ of one piece based on the rotation and position of the other.

To compute the values of the dependencies we use the same method as in the chapter 5.1.2 to compute the optimal geometric layout of two edges applied on the adjacent edges of two pieces.

The optimal local dependency between the frame and the piece having an edge without padding is slightly more complicated and depends on the part of the frame the piece is coincide with. The frame is represented by an axis-parallel rectangle with lower left corner located in the point $(0,0)$ and the unknown width w and height h .

The dependencies are always stating the ideal absolute value ω'_i of each piece's rotation. Moreover for the pieces coinciding with left or bottom edge we define their ideal absolute value of x'_i or y'_i . On the other hand, for pieces coinciding with the right or top frame we define their relative value x''_i or y''_i compared to the unknown width or height of the frame.

We compute the dependency between the frame pieces and the frame similarly as in the piece to piece dependency case considering the frame as a flat edge.

The computed dependencies provide an insight into the relations between values in a virtually ideal, maximally coherent configuration. However this configuration could not exist, it allows us to find the solution closest to this ideal state.

6.1.2 Configuration based on dependencies

The computation of the final configuration is based on the computed local dependencies. The computation is done in two steps, first we compute the rotation value of each piece ω_i and then we compute the translation of the pieces x_i, y_i together with dimension of the puzzle w, h .

In both steps, we transform the problem into an instance of a overdetermined linear system and find the solution using the least square method.

computing of rotations

The computing of each piece's rotation directly uses obtained dependencies as the equations of the linear system having variables ω_i :

$$\omega_i - \omega_j = \Delta\omega_{ij} \quad \forall i, j \text{ adjacent pieces} \quad (6.1)$$

$$\omega_i = \omega'_i \quad \forall i \text{ border piece} \quad (6.2)$$

Before solving the system we need to consider the 2π -cyclic nature of the angular values. The proposed linear system can not handle this directly causing wrong results. To overcome this problem, first we rotate each piece in an approximately correct direction based on the edge facing up and then apply the linear system to compute the exact values.

computing of translations

The computing of each piece's translation with dimension of the puzzle is done similarly using the obtained dependencies solving the linear system:

$$x_i - x_j = \Delta x_{ij} \quad \forall i, j \text{ adjacent pieces} \quad (6.3)$$

$$y_i - y_j = \Delta y_{ij} \quad \forall i, j \text{ adjacent pieces} \quad (6.4)$$

$$x_i = x'_i \quad \forall i \text{ piece on the left border} \quad (6.5)$$

$$x_i - w = x''_i \quad \forall i \text{ piece on the right border} \quad (6.6)$$

$$y_i = y'_i \quad \forall i \text{ piece on the bottom border} \quad (6.7)$$

$$y_i - h = y''_i \quad \forall i \text{ piece on the top border} \quad (6.8)$$

Note, the values Δx_{ij} , x'_i , y'_i , x''_i , y''_i are also rotated based on the computed values of ω_i .

6.2 Image composition

The computed configuration of the puzzle is transformed into resulting image of the assembled puzzle.

First, we create an image of the computed size and then we place the pieces into this image one at a time. During the processing of one piece, we cut the piece from the source image, apply the particular transform and merge it with the image.

We can also apply the morphologic operation erode[19] on each piece to improve the quality of the resulting image. Using the circle kernel of size a few pixels eliminates errors caused by non-perfect cutting of the piece from its source image. This also improves the result in the occasional partial overlap of neighbouring pieces.

The proposed visualization of the computed configuration involving drawing the pieces is the final an termination step of the entire algorithm.

Chapter 7

Implementation and optimization

This chapter describes the various aspects of the implementation of the proposed method. This area is becoming especially important when we use the method on systems with limited computing resources.

First we describe a parallelisation possibility of the method. Parallelisation is an important aspect dramatically reducing the running time on systems having multiple computational units.

Then we describe the possible optimizations of the most critical parts of the method. This parts have the greatest impact on the overall running time and their optimization is the important aspect of the possible computational time reduction.

The stated improvements provide a practical insight into an effective implementation of the method and provide the effective solutions for the most computational complex parts.

7.1 Parallelisation

The method can be very effectively scaled in terms of the parallelisation if we have the sufficient number of parallel execution units. This is based on the high scalability of each of the most important parts of the method.

First, the pieces are extracted from independent image pairs during the data extraction phase. This process can be naturally parallelised processing each input image using different execution thread. The parallelisation scale is therefore limited only by the number of input images.

Second, the computing of compatibility table can be also handled very effectively. We can split the process to several threads, each one computing one row of the table or even only part of the row. This allows almost unlimited boost scale on the systems having many execution units.

Next, the solving of the interior of puzzle; in each step we need to examine several possibilities where and which piece could be placed. Each this possibility can be processed in the separate thread making the process very fast.

Finally, the process of puzzle visualisation when the solution of overdetermined linear system is computed can be parallelised as well. This can be achieved using one of many known parallel implementations of the used least squares method.

All this improvements together can dramatically reduce the running time from possibly several hours to several minutes if the sufficient computing power is available. This speed then outperforms any attempt to solve the given puzzle by a traditional hand-solving method.

7.2 Special algorithms

The method consists of two computationally demanding parts. We propose the possibility of effective optimization of this parts to decrease the computational time of the method.

First, we describe a technique which can significantly boost up the Shape optimization algorithm defined in the section 4.2.2. Then we propose a effective way of computing the compatibility table used in the chapter 5.

7.2.1 Shape optimization algorithm

The described shape optimization algorithm requires iterative reoptimization process of the shape transformation involving routine computation of the mass center of the given square neighbourhood. We propose a method, which can after linear preprocessing handle each this query in a constant time rapidly boosting the entire algorithm.

We precompute two 2D tables of the size equal to the size of the image *width* and *height* whose square areas we ask the queries. One table is for *x*-dimension and the values are $S(x, y)$ and the second is for *y*-dimension with values $R(x, y)$.

Let denote the pixel's edge value assigned by the edge detection as $W_{x,y}$, the values of the tables are defined by:

$$S(x, y) = \sum_{i=0}^{x-1} \sum_{j=0}^{y-1} iW_{i,j} \quad (7.1)$$

$$R(x, y) = \sum_{i=0}^{x-1} \sum_{j=0}^{y-1} jW_{i,j} \quad (7.2)$$

The closed form of the cell values can be derived as:

$$S(x, y) = S(x - 1, y) + S(x, y - 1) - S(x - 1, y - 1) + iW_{x-1,y-1} \quad (7.3)$$

$$R(x, y) = R(x - 1, y) + R(x, y - 1) - R(x - 1, y - 1) + jW_{x-1,y-1} \quad (7.4)$$

If we fill the tables by a row order starting from the lower left corner, we always have the values in the closet form already computed. This allows us to spend a constant time with computing of one value finishing the entire computation in a linear time.

Next, we can effectively handle the queries. Each query is asking for a center of mass of some square with center point p and radius r . We can translate this as a query asking for the center of mass of the rectangle defined by two points $a = (p_x + r, p_y + r)$ and $b = (p_x - r, p_y - r)$. Answer for each this rectangle query can be computed as a point defined as:

$$c_x = S(a_x + 1, a_y + 1) - S(b_x, a_y + 1) - S(a_x + 1, b_y) + S(b_x, b_y) \quad (7.5)$$

$$c_y = R(a_x + 1, a_y + 1) - R(b_x, a_y + 1) - R(a_x + 1, b_y) + R(b_x, b_y) \quad (7.6)$$

Described approach of the linear precomputation of the table and later processing of arbitrary queries in the constant time speeds up the algorithm by about order of magnitude allowing us to decrease the running time of the data extraction process.

7.2.2 Computing of compatibility table

The computation of the compatibility table requires computing of the compatibility score for every pair of logically compatible edges. The entire process requires to determine the optimal shape align for every pair of the edges which is a relatively expensive operation. The observation the high compatibility score is achieved only by some subset of the entire edge pairs set allows the effective optimization.

The edge pairs which have the low compatibility score never coincide in the correct solution and they are not used by the algorithm during the solving. It is enough if we only roughly determine their theoretical score indicating the incompatibility. The problem is, we can not guess which edges have the high and low compatibility score without computing it.

To resolve this situation, we create for every edge several versions in a lower resolution. In each this lower resolution is every edge defined by a less number of points. Then we continue in several rounds, for every resolution one.

First we run the computation of the scores on the lowest resolution for every pair of edges. In every next round we recompute the score for some fraction of the best scoring edge pairs. The chosen fraction should reflect the amount of this edges in the dataset. In the last round is therefore examined the score for the best matching edges in a full resolution obtaining precise results.

The approach allows us to distribute the computation based on the final compatibility score therefore optimizing the running time. The score for compatible edge pairs is determined precisely and for incompatible edge pairs is the score determined with less precision.

Chapter 8

Experimental results

To measure the properties and abilities of the proposed method, we wrote the implementation and experimentally tested its abilities. The testing involved several aspects.

The implementation was written in the C++ language and can be found in the attachment. We choose this language due to its high efficiency of the compiled code and a low memory overhead. These proprieties are becoming especially valuable during solving of big puzzles.

The implemented method was tested on inputs of various size and properties. We choose the puzzles with different colour variability and size starting on puzzles having 80 pieces up to puzzles having over 1000 pieces.

We tested several aspects of the method. The most important one was the ability to successfully solve the given puzzle. Another aspect was the testing of this ability on data having lower resolution and under different settings of the method. We also measured the total running time and memory requirements during the solving.

The testing showed different properties of the method and the overall performance on given data. If the quality of the input data was good enough, the method was able to successfully solve all given puzzles (Figure 11.4, 11.5, 11.6, 11.7).

8.1 Choice of testing data

The input data were the scanned pieces of standard puzzles which can be bought in almost every local toyshop.

The testing was done on the puzzles by Ravensburger AG. This puzzle have a high quality and contain less physical errors like erosion of the pieces. The size of the puzzle was chosen 80, 200, 572 and 1008 pieces. Puzzle had different themes and variability of the texture and colour.

The puzzle were scanned on a standard scanner Genius ColorPage HR7X Slim in the resolution 100 dpi using the format described in chapter 4. This resolution produces images of pieces having the edge length about 150 pixels

which is sufficient for precise extraction of the shape. The contrast and brightness were set to a constant value during the entire scanning to values providing a good segmentation of the image into pieces and background.

We also created input data with digitally decreased resolution of the original data to the lower values 75, 50 and 25 dpi. The testing on this data measured a dependency of method's successfulness on the quality of input.

8.2 Performance

The performance of the method was tested on the personal computer Lenovo T400 in a configuration Intel Core2 Duo 2.4 GHz, 4GB RAM.

The running time and memory requirements on different inputs when the method used one execution thread:

size of puzzle	running time (min)	required memory (MB)
80 (100dpi)	4	21
80 (75dpi)	3	17
80 (50dpi)	3	14
80 (25dpi)	2	9
200 (100dpi)	9	52
200 (75dpi)	8	46
200 (50dpi)	6	42
200 (25dpi)	-	-
575 (100dpi)	29	124
575 (75dpi)	24	117
575 (50dpi)	17	109
575 (25dpi)	-	-
1000 (100dpi)	111	405
1000 (75dpi)	94	396
1000 (50dpi)	-	-
1000 (25dpi)	-	-

- means the method did not solve the given input in time 10 hours.

To compare the parallelisation efficiency we also tested the method with two execution threads on the inputs with resolution 100 dpi:

size of puzzle	running time (min)	required memory (MB)
80 (100dpi)	3	21
200 (100dpi)	6	52
572 (100dpi)	16	55
1008 (100dpi)	71	412

8.2.1 Robustness

The results indicates the performance of the method is dependant on the quality of input data. However the method is able to solve the puzzles of size 1008 pieces even on the low resolution 75 dpi of input data.

It is required to have a higher resolution of input data to successfully solve the puzzles of bigger size than for the puzzles with a lower number of pieces.

The very low resolution causes a insufficient precision of the extracted shape and colour. This makes difficult to distinguish matching and non-matching edges based on their compatibility score. The method can then fail because of finding the wrong solution of the frame or the wrong filling of the interior of the frame.

8.2.2 Shape-only case

To measure the importance of the shape and colour information we tested the method in a configuration using only the shape information. This approach is similar to previous methods which were concentrated exclusively on using only this type of information.

Using the shape information only was achieved by setting the coefficients defined in the section 5.1.3 to the values $\alpha = 1, \beta = 0, \gamma = 0, \delta = 0$. Testing showed, the method was able to successfully solve all tested puzzles on resolution 100 dpi and some of them also on lower resolutions. However, in most of the cases, the solving of the frame required more iterations than in the case of using both the shape and colour information.

This experimental testing pointed on the fact, the using of shape information only is enough to successfully solve even very large puzzles.

8.2.3 Running time

The stated results provide the insight of the method's efficiency measured in the time necessary to solve the particular puzzle.

The testing stated approximately quadratic dependency of the running time based on the number of pieces. This dependency is mostly given by the compatibility table computation phase. In this phase is computed the score for every pair of edges.

On the other hand, the puzzle frame solving phase as the only one with the theoretical exponential complexity was usually done in a few iterations of the algorithm and did not affected the running time of the method.

The dependency of the running time based on the resolution is mostly linear. This is based on the fact, the majority of the used algorithms is linearly dependant on the length of piece's edges.

The running time in configuration with two execution threads was approximately half of the running time with one execution thread. This showed a very good parallelisation capability of the method.

In conclusion, the experimental analysis showed the running time of the method is mostly based on the resolution of the inputs and the size of the puzzle.

8.2.4 Memory requirements

The size of operating memory required by the method is dependant on several factors.

Memory requirements shows approximately quadratic dependency based on the number of pieces. This fact is based on structures used to store the data, especially on the table storing the compatibility score for every pair of edges.

On the other hand, a constant information in this table for every pair of edges caused that the resolution of input data does not have a big impact on the size of required memory. Slightly lower memory requirements were based on a less amount of the information describing the pieces itself. This information contains the shape and the colour description of pieces along the edges and its size is therefore dependant on the length of the edges defined by the used resolution.

Overall memory requirements were therefore dependant mostly on the size of the puzzles.

Chapter 9

Discussion

The algorithm itself achieves very good results and solves the given problem successfully. Moreover some extensions and applications can be made.

The proposed method can be extended in several ways. One of them is ability to solve puzzles of arbitrary shape similarly to work [6]. Another interesting extension could be application of the method during designing of a robotic system capable to mechanically solve the real puzzle.

The developed algorithms and techniques can be applied in other areas as well. For example the fast method used for determining the optimal layout of two matching edges can be used in applications of noisy curves matching. Next, the method of matching the shape of the piece on its image can be used in areas where the using of Hough transform is impractical or impossible.

To sum up, the method can be further extended to greater capability. Moreover the developed algorithms touch some connected areas and the results can be applied in this areas as well.

Chapter 10

Conclusion

In this thesis we introduced a new algorithm for the automatic solution of digitalised jigsaw puzzle. In many areas this algorithm uses more sophisticated methods than previous work. The algorithm consisted on several consecutive phases.

First, we showed how parameters of scanned pieces can be precisely extracted. This is the the base aspect of the successful solving of the puzzles. Next, we used an advanced method of measuring piece-wise similarity. In this method we used a score-based on the shape and colour compatibility of optimal geometric layout of given pieces. During the assembling of the pieces we used a two-step algorithm of solving the frame first by reducing the problem to the max-cost bipartite matching and solving the interior second using the greedy algorithm with limited backtracking. Finally we showed the method of solution visualisation into a highly coherent image.

We suggested how the method can be effectively implemented and parallelised and also provided the implementation in C++. Testing showed this method was able to solve the puzzles of greater size than all previous approaches when it solved the puzzle of size more than 1000 pieces. Results also indicated the success of the method depends on the quality and resolution of input data especially on the quality of shape information.

At the end we discussed the possible extensions and application of the method and the specially developed algorithms for this task.

Bibliography

- [1] H. Freeman and L. Garder. "Apictorial jigsaw puzzles: the computer solution of a problem in pattern recognition". IEEE TEC, (1964) (13): pp. 118–127
- [2] Radack, Gerald M. and Norman I. Badler. "Jigsaw puzzle matching using a boundary-centered polar en- coding," Computer Graphics and Image Processing, VOI 19. (1982), pp. 1-17
- [3] H.E. Wolfson, A. Schonberg, A. Kalvin, Y. Lamdan. "Solving Jigsaw Puzzles by Computer," Annals of Operation Research, 12 (1988) pp. 51-64
- [4] D.A. Kosiba, P.M. Devaux, S. Balasubramanian, T. Gandhi, R. Kasturi. "An Automatic Jigsaw Puzzle Solver". Int. Conf. Pattern Recognition, Jerusalem, (1994), pp. 616–618.
- [5] M.G. Chung, M. Fleck and D.A. Forsyth. "Jigsaw Puzzle Solver Using Shape and Color". ICSP '98, (1998), pp. 877–880.
- [6] D. Goldberg, C. Malon, and M. Bern. "A global approach to automatic solution of jigsaw puzzles". In Symposium on Computational Geometry, (2002).
- [7] T. R. Nielsen, P. Drewsen, and K. Hansen. "Solving jigsaw puzzles using image features". PRL, (2008).
- [8] L. Atsmon L. Varon, "Automatic Jigsaw Puzzle Solver" thesis. Israel Institute of Technology, (2009).
- [9] T.S. Cho, S. Avidan and W.T. Freeman. "A probabilistic image jigsaw puzzle solver" Massachusetts Institute of Technology, (2010).
- [10] G.C. Burdea and H.J. Wolfson. "Solving Jigsaw Puzzles by a Robot". IEEE Trans. on Robotics and Automation 5 (1989), pp. 752–764.
- [11] E. D. Demaine and M. L. Demaine. "Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity". Graphs and Combinatorics, (2007).

- [12] Duncan Richer, "The Eternity Puzzle" (July 1999)
- [13] Applegate, D. L.; Bixby, R. E.; Chvátal, V.; Cook, W. J. "The Traveling Salesman Problem: A Computational Study", Princeton University Press, ISBN 978-0-691-12993-8. (2006)
- [14] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. "Introduction to Algorithms (second ed.)". MIT Press and McGraw-Hill. ISBN 0-262-53196-8 (2001) Chapter 26, pp. 643–700.
- [15] Assignment problem Burkard, Rainer; M. Dell'Amico, S. Martello. "Assignment Problems". SIAM. ISBN 978-0-898716-63-4 (2009).
- [16] Katta G. Murty. "An algorithm for ranking all the assignments in order of increasing cost", Operations Research, Vol. 16, No. 3, (May-June, 1968), pp. 682 – 687
- [17] MacQueen, J. B. "Some Methods for classification and Analysis of Multivariate Observations". 1. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press. (1967) pp. 281–297
- [18] Nobuyuki Otsu. "A threshold selection method from gray-level histograms". IEEE Trans. Sys., Man., Cyber. 9 doi:10.1109/TSMC.1979.4310076 (1979) (1): pp. 62–66.
- [19] Jean Serra. "Image Analysis and Mathematical Morphology", ISBN 0126372403 (1982)
- [20] Mark S. Nixon and Alberto S. Aguado. "Feature Extraction and Image Processing". Academic Press, (2008), p. 88.
- [21] Duda, R. O. and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," Comm. ACM, Vol. 15, (January, 1972) pp. 11–15
- [22] Schwartz J.T. and Sharir M. "Identification of partially Obscured Objects in Two Dimensions by Matching of Noisy Characteristic Curves", Tech. Rep. No.165, Comp. Sc. Div., Courant Inst. of Math., NYU (June 1985)
- [23] Rafael C. Gonzalez and Richard Eugene Woods. "Digital Image Processing", 3rd ed. Upper Saddle River, NJ: Prentice Hall. ISBN 0-13-168728-X. (2008) pp. 407–413.
- [24] Bretscher, Otto. "Linear Algebra With Applications", 3rd ed. Upper Saddle River NJ: Prentice Hall (1995)

Chapter 11

Attachment

11.1 Source code and testing data

The implementation of the algorithm together with testing data are located on the enclosed compact disc with the directory structure:

./src	implementation source code in C++ language
./doc	documentation of the source code
./bin	a compiled binary for Unix system
./input	input data for the puzzles of size 80, 208, 572 and 1008
./output	solved puzzles of size 80, 208, 572 and 1008
./thesis	an electronic version of this thesis

11.2 Example input 1

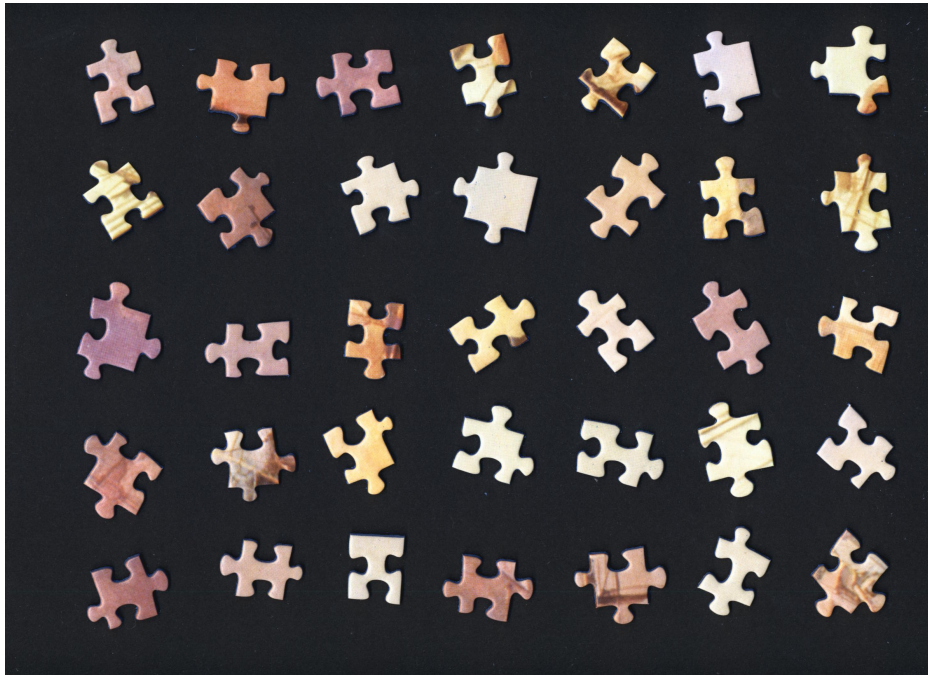


Figure 11.1: Input image with the front sides of bright puzzle pieces scanned against the dark background.

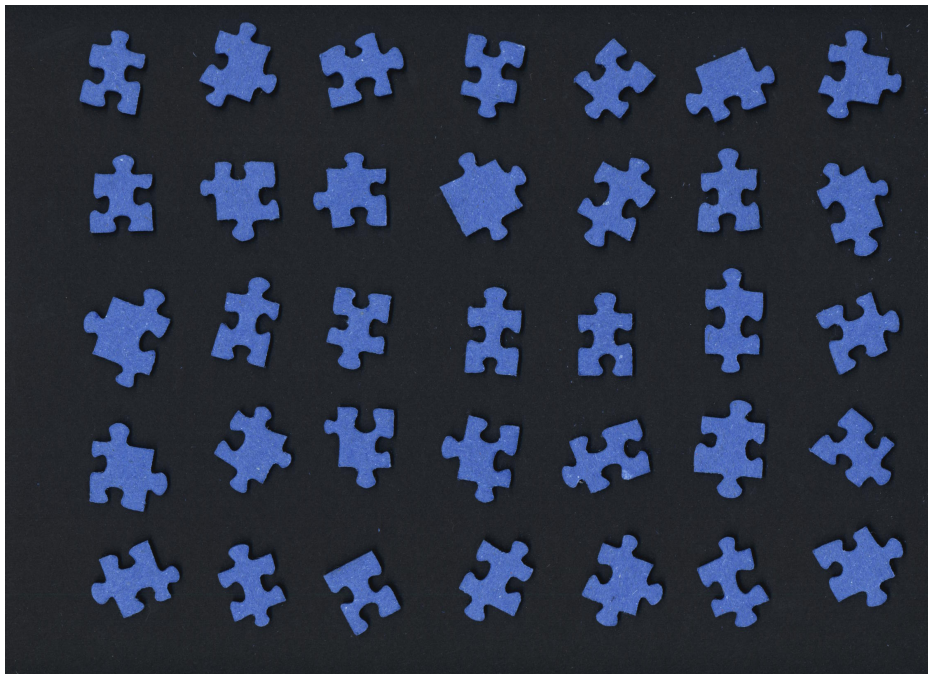


Figure 11.2: Corresponding image with the back sides of pieces.

11.3 Example input 2

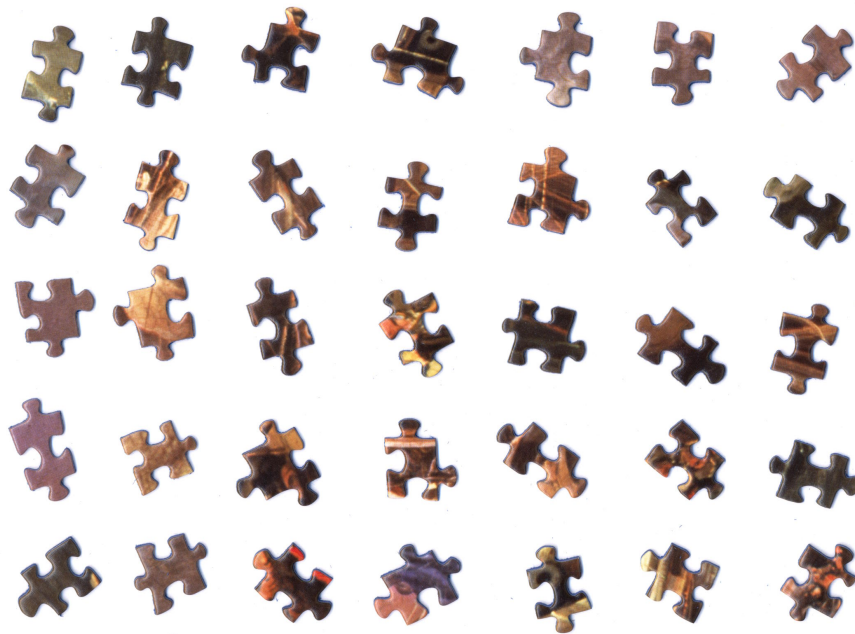


Figure 11.3: Input image with the front sides of dark puzzle pieces scanned against the bright background.

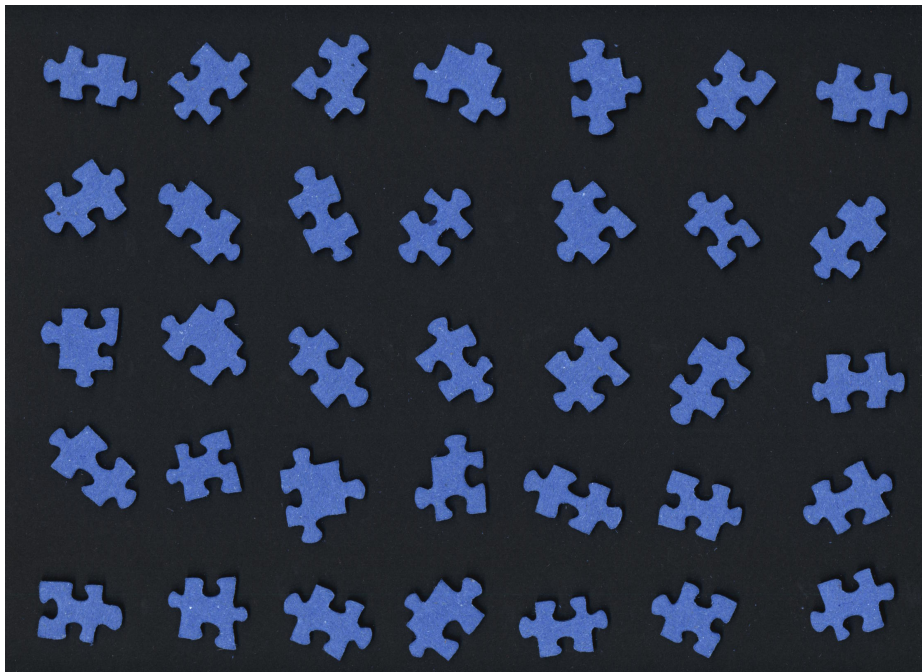


Figure 11.4: Corresponding image with the back sides of pieces.

11.4 Solution to a 80-piece puzzle



Figure 11.5: Output image with the solution of the 80-piece puzzle.

11.5 Solution to a 208-piece puzzle

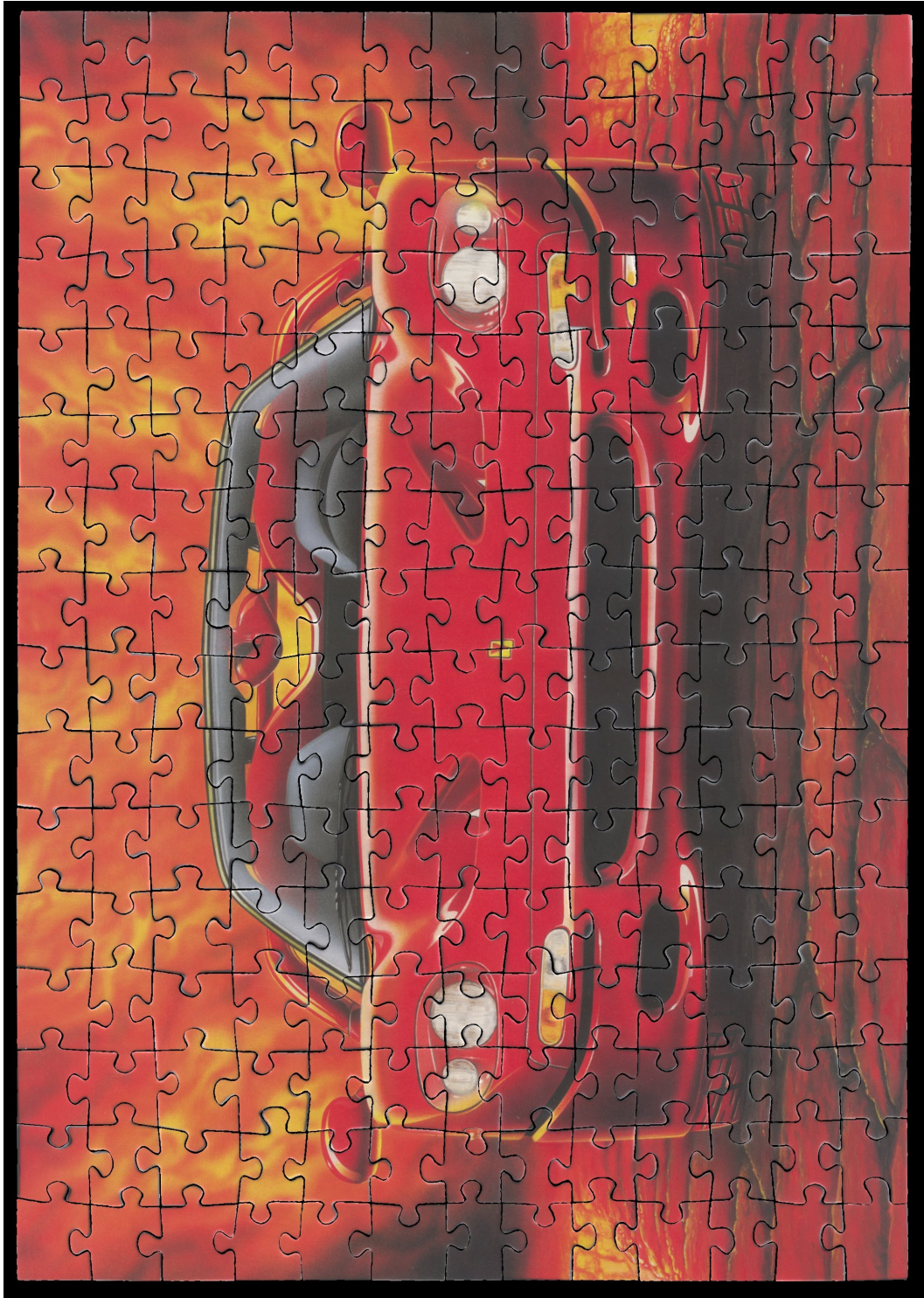


Figure 11.6: Output image with the solution of the 208-piece puzzle.

11.6 Solution to a 572-piece puzzle

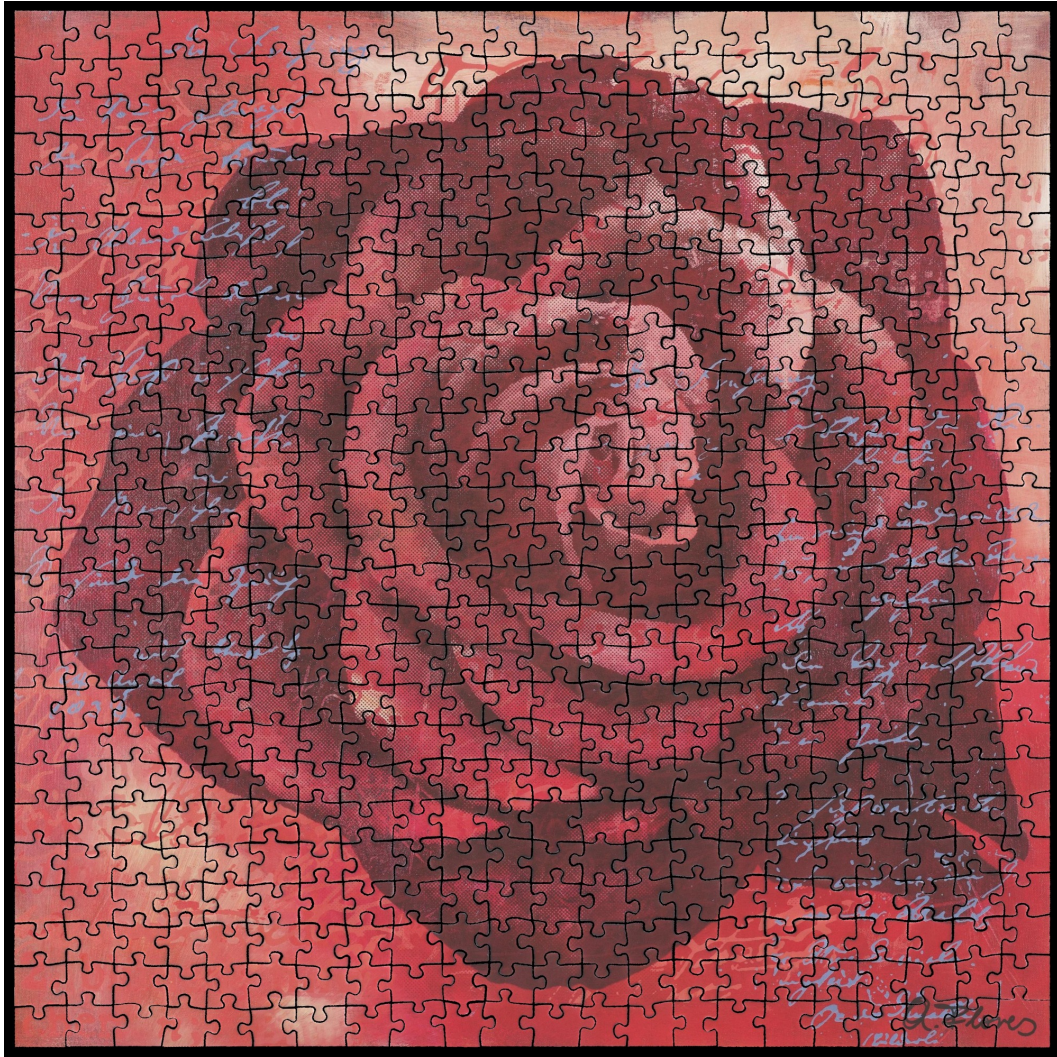


Figure 11.7: Output image with the solution of the 572-piece puzzle.

11.7 Solution to a 1008-piece puzzle



Figure 11.8: Output image with the solution of the 1008-piece puzzle.